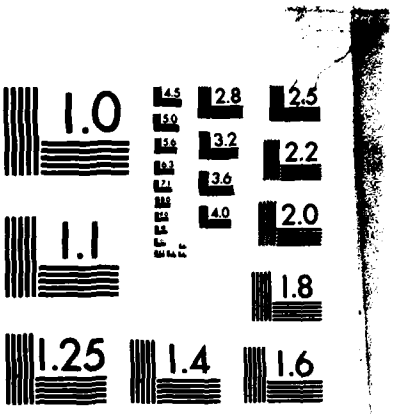
AD-A131 775 DACS (DATA AND ANALYSIS CENTER FOR SOFTWARE) DATA COLLECTION PRODUCTIVITY FORMS(U) DATA AND ANALYSIS CENTER FOR SOFTWARE GRIFFISS AFB NY MAR 79 1/1

UNCLASSIFIED F30602-78-C-0255 F/G 5/2 NL

END

FILMED

DTIC



SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO. AD-A131 775

3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*
DACS DATA COLLECTION PRODUCTIVITY FORMS

5. TYPE OF REPORT & PERIOD COVERED
Interim Report
Nov. 1978 - March 1979

6. PERFORMING ORG. REPORT NUMBER
N/A

7. AUTHOR(s)
N/A

8. CONTRACT OR GRANT NUMBER(s)
F30602-78-C-0255

PERFORMING ORGANIZATION NAME AND ADDRESS
Data & Analysis Center for Software
RADC/ISISI
Griffiss AFB, NY 13441

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

CONTROLLING OFFICE NAME AND ADDRESS
Rome Air Development Center (COEE)
Griffiss AFB, NY 13441

12. REPORT DATE
March 1979

13. NUMBER OF PAGES
11

MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*
Same

15. SECURITY CLASS. *(of this report)*
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Same

18. SUPPLEMENTARY NOTES
RADC Project Engineer: John Palaimo (COEE)
DACS Source Code No. 413570
Cost: No Charge

Available from:
Data & Analysis Center for Software
RADC/ISISI
Griffiss AFB, NY 13441

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*
Software Experience Data
Programmer Productivity
Data Repository
Data Parameters

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*
Two data collection forms were designed for software life cycle productivity-related data. The first form, the Project Summary Form, is used to describe the project's software development environment. The second form is the Component Summary which is used to describe the software components of a project, resources expended during its development, and any noted discrepancies. Component data may be collected at several levels such as a System, Functional Group, and Module. In general, the forms have been designed to identify key data collection areas but are flexible enough to accommodate additional project-

(over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

specific data. The set contains 2 forms and 8 pages of definitions and guidelines for use of the forms.

DACS DATA COLLECTION PRODUCTIVITY FORMS

Introduction

DACS has designed two data collection forms for software life-cycle productivity-related data. The first form, the Project Summary Form, describes a project's software development environment. The second form is the Component Summary and describes the software components of a project, resources expended during its development, and any noted discrepancies. Component data may be collected at several levels such as System, Functional Group, and Module. Figure 1 describes the general attributes and relationships of each component level. In general, the forms have been designed to identify key data collection areas yet be flexible enough to accommodate additional project-specific data.

Project Summary Form

This form is used to collect data concerning (1) the general nature of the project, (2) the project priorities and constraints, and (3) the software development technologies used in the project. The data elements of project name, project type, project description, number of systems, number of functional groups, number of modules and standards record the general nature of the project. Priorities and constraints may be chosen from several common ones listed on the form, or project-specific entries may be noted. A list of software development technologies is also provided to which the project may add entries.

Component Summary Form

This form is used to collect data concerning (1) the software engineering characteristics of the component, (2) the resources expended throughout the life cycle of the component and (3) the discrepancies detected in the component's life cycle. Complexity, type, description, size, programming language, and mode of construction describe the component in software engineering terms. The resources expended on the component are recorded by life-cycle phase and time in terms of person-hours and computer-hours. The number of discrepancies found in the component are recorded by life-cycle phase.

Purpose

The purpose of these forms is to collect project and component level data that is common to most software development projects. If these forms are to be used as a tool for collecting data for a specific quantitative software engineering model or method, additional project-specific forms should be developed. To develop these special forms, DACS Software Engineering Research Review - Quantitative Software Models, SRR-1, should be used to determine the data parameters to be collected for the specific application.

DACS DATA COLLECTION FORM

PROJECT SUMMARY

INSTRUCTIONS

The purpose of the project summary form is to describe the project development environment as the project evolves through the software life-cycle. It should be completed by the project manager at the beginning of the project, at each major milestone, and at the end. Data at the initiation of the project are estimated; intermediate reports should change estimates to actuals and update estimates, and the final report should describe the actual project environment.

DEFINITIONS

Explanations and definitions of the form's data elements are presented below.

1. Reporting Period - Indicate the time period (start and end date) and life-cycle phase.

2. Project Name - Name of the project.

3. Description - An overview of the mission of the project.

4. Project Duration - Total length of the project in months.

5. Number of Systems - A system is a software component that provides a meaningful product to the user and is usually capable of operating independently of other systems. Indicate the number of systems in the project software.

6. Number of Functional Groups - A functional group is a software component that satisfies a set of functional and performance specifications. Indicate the number of functional groups in the project software.

7. Number of Modules - A module is a software component that exists as a discrete, identifiable set of instructions. Indicate the number of modules in the project software.

8. Contact - Person completing the form.

9. Developer - Name of the organization responsible for delivery of the project.

10. Staff Size - List the number of persons assigned to the project. Include administrative, technical and clerical personnel.

11. Contract - Contract number.

12. Management Organization - A short description of the project's management structure.

13. Cost Reporting Standards - Standards, specifications or methods used to report project costs.

14. Technical Standards - Standards, specifications or other requirements related to the technical aspects of the project such as requirement specification, coding, testing, documentation, configuration management and quality assurance.

15. Constraints - Note the constraints applicable to the project. Constraints not listed may be added.

16. Computers - List the hardware and operating system configurations of the development computer (system on which the software was developed) and the target computer (system for which the software was developed). Also, list the type of access to each computer (e.g., batch, remote batch).

17. Documentation - Delivered pages of documentation. Includes program listings, flow charts (low and hi-level), operating procedures, maintenance procedures and any other descriptive material covering the design, development, test, operation, installation and maintenance of the software.

18. Priorities - Note the priorities applicable to the project. Priorities not listed may be added.

19. Technology - Note the software engineering technologies and techniques applicable to the project. Technologies not listed may be added. Attachment 1 defines the listed technologies.

    % Utilization - Percent of the total lines of source code developed using this technology.

20. Support Software Developed by Project - List the software that was developed by the project to specifically support the development of the project deliverable software. Software in this category varies widely but typically includes test generators, test stubs, test drivers, assemblers, compilers, and simulators.

* Parameter common to both RADC Productivity Database and NASA-SEL Database.

† Parameter necessary for RADC Productivity Database only.

DACS

Data & Analysis Center for Software

DATA COLLECTION FORM

PROJECT SUMMARY

Reporting Period: _____

Project Name*: _____

Description*: _____

_____

Project Duration*: _____

Number of Systems: _____

Number of Functional Groups: _____

Number of Modules*: _____

Contact: _____

Developer: _____

Staff Size: _____ Contract _____

Management Organization: _____

Cost Reporting Standards: _____

Technical Standards:

_____ _____ _____

_____ _____ _____

Constraints:

( ) Hardware Limitations
( ) Limited Schedule
( ) Limited Computer Accessibility
( ) Limited Funding
( ) Limited Staffing
( ) Limited Management Support
( ) Target Computer Different
( ) New Application
( ) CPU/Memory Constraint
( ) CPU/Time Constraint
( ) _____
( ) _____

Computers:

	Hardware	Oper. System	Access
Development	_____	_____	_____
Target	_____	_____	_____

Documentation†: _____

Priorities:

( ) Proc. Speed
( ) Core Utilization
( ) Schedule
( ) Cost
( ) Quality
( ) Reliability
( ) _____
( ) _____
( ) _____

Technology*:

	% Utilization†		% Utilization†		% Utilization†
( ) Chief Programmer Team	____	( ) Program Support Library	____	( ) High Order Language	____
( ) Automated Design Tools	____	( ) Simulator	____	( ) Pre-compiler	____
( ) Automated Requirements Tools	____	( ) Structured Programming	____	( ) Reusable Code	____
( ) HIPO Design Aides	____	( ) Walk-throughs	____	( ) Correctness Proofs	____
( ) Process Design Language	____	( ) Critical Piece First	____	( ) Code Standards Auditor	____
( ) Structure Charts	____	( ) Database Analyzer	____	( ) Consistency Checker	____
( ) Top-Down Development	____	( ) Data Dictionary	____	( ) Independent Test Team	____
( ) Modular Decomposition	____	( ) Documentation Generator	____	( ) Program Flow Analyzer	____
( ) _____		( ) _____		( ) _____	
( ) _____		( ) _____		( ) _____	

Support Software Developed by Project: _____ _____ _____

_____ _____ _____

DACS DATA COLLECTION FORM

COMPONENT SUMMARY

INSTRUCTIONS

The purpose of the component summary form is to describe the component's structure, resources used during its life cycle and problems reported during its life cycle. The form should be filled out for each component at the completion of each phase by the person responsible for components. The summary may be used for the module, functional group, and system component levels.

DEFINITIONS

Explanations and definitions of the form's data elements are presented below.

1. Reporting Period - Indicate the time period (start and end date) and life-cycle phase.

2. Component Name - Name of the component.

3. Component Type - System, functional group or module. A system is a software component that provides a meaningful product to the user and is usually capable of operating independently of other systems. A functional group is a software component that satisfies a set of functional and performance specifications. A module is a software component that exists as a discrete, identifiable set of instructions. (See Figure 1 for the hierarchical relationships between the components.)

4. Description - An overview of the component, including its purpose.

5. Contact - Person completing the form.

6. Mode of Construction - List the modes of construction for the component.

   Modular - Components are constructed such that each component has single entry and exit points and a single purpose.

   Unstructured - The type of programming constructs is not limited to a specific set. (See Structured).

   Top-Down - The component is constructed by identifying major functions to be accomplished and then proceeds from there to an identification of the detail functions that derive from the major ones.

   Structured - The component is constructed using a limited set of programming constructs. (E.g., SEQUENCE, DOWHILE, DOUNTIL, CASE, IFTHENELSE.)

   Conventional - Structured and/or top-down constructions are not used.

7. Programming Language - List the programming languages used for the component and the percent utilization of each. Base the percent utilization on the number of lines of source code.

8. Component Complexity -

   Subjective - Rate the subjective complexity of the module as easy, moderate, or difficult.

   Calculated - List any calculated complexity metrics (e.g., number of decision-to-decision paths) that have been determined for the component.

9. Component Size - Categorize the component into new code (developed for this application) and reused code (developed for another application but used in this application) and indicate its size by source instructions (number of source code lines including comments) and object words (computer words).

10. Resources - List the person-hours (divided into technical and administrative tasks) and computer-hours expended in development of the component for each of the life-cycle phases listed below.

Phase	Description
Conceptual	Defining the problem statement preliminary systems analysis and identifying alternative solution strategies.
Requirements	Producing statement of project objectives, system functional specifications and design constraints.
Design	Generalizing software component definitions, interfaces and data definitions; verifying these against requirements.
Implementation	Generating program code, testing code, and documenting the system.
Test	Integrating software components and performing system acceptance tests.
Operation	Using and maintaining the system.

11. Reported Problems - List the number of problems reported in the component during the life-cycle phases and the method used to detect the problem. For example, a typical detection method used during the design phase is the formal design review; for the test phase the detection method might be the system integration test.

DACS

Data & Analysis Center for Software

DATA COLLECTION FORM

COMPONENT SUMMARY

Reporting Period: _____ _____ **Contact:** _____

Component Name: _____

Component Type: _____

Description: _____

_____

Mode of Construction*: ( ) Modular ( ) Structured ( ) Unstructured ( ) Conventional ( ) Top-down _____

_____ _____

Programming Language*:

Name	% Utilization
_____	_____
_____	_____
_____	_____

Component Complexity:

Subjective	Calculated Type	Calculated Measure
( ) Easy	_____	_____
( ) Moderate	_____	_____
( ) Difficult	_____	_____
	_____	_____
	_____	_____
	_____	_____

Component Size*: *

	New	Reused
Source Instructions	_____	_____
Object Words	_____	_____

Resources:

	Person-hours* Tech/Admin	Computer-hours
Conceptual	_____	_____
Requirements	_____	_____
Design	_____	_____
Implementation	_____	_____
Test	_____	_____
Operation	_____	_____
Total*	_____	_____

Reported Problems:

	Detection Method	Number*
Conceptual	_____	_____
Requirements	_____	_____
Design	_____	_____
Implementation	_____	_____
Test	_____	_____
Operation	_____	_____
Total*	_____	_____

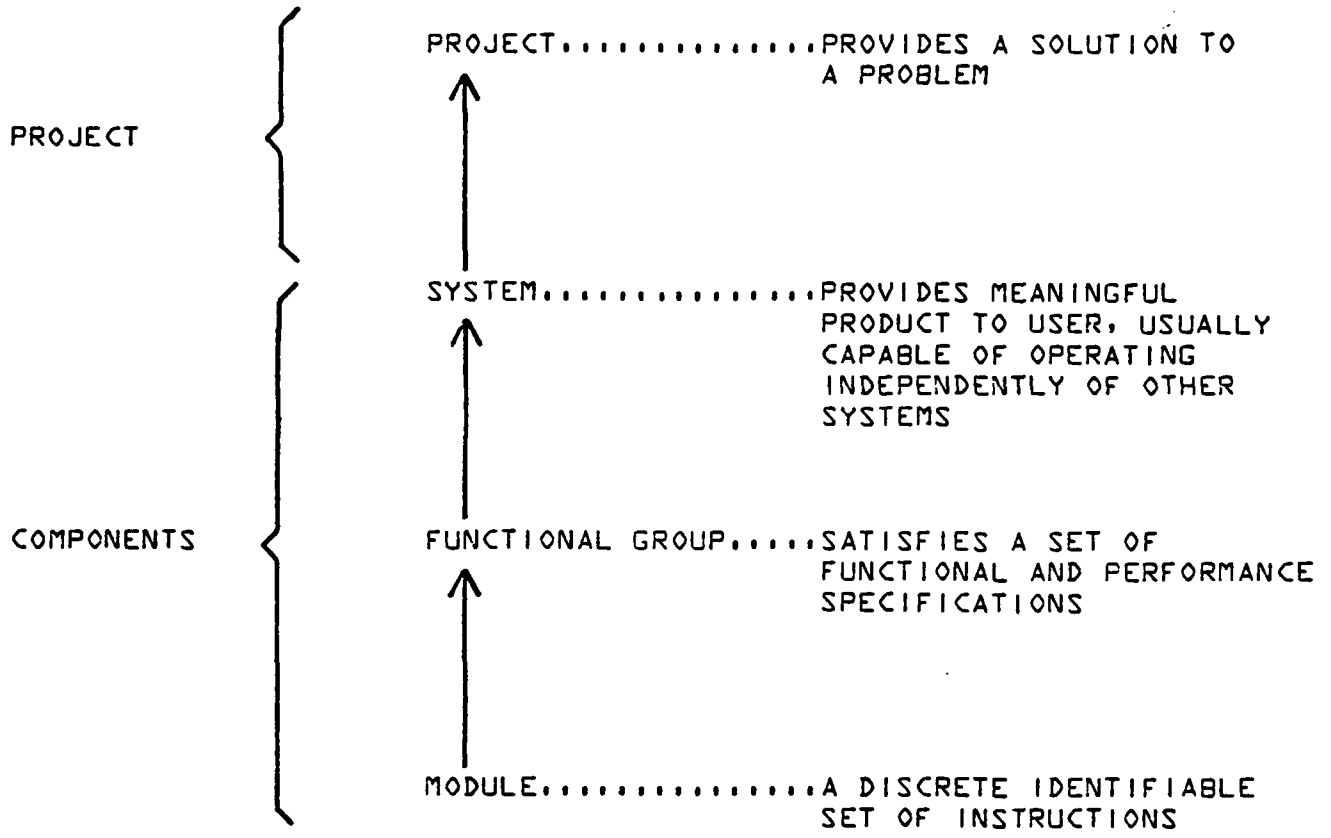


Figure 1. Project-Component Attributes and Relationships

APPENDIX 1

TECHNOLOGY DEFINITIONS

1. Chief Programmer Team - A chief programmer team is a structured team of specialists for software development headed by a chief programmer. The team has at its core three members: the chief programmer, the back-up programmer and the secretary/librarian. The rest of the team consists of programmers as required. The team is normally limited to less than ten members.

2. Automated Design Tools - Computer programs used to provide an understandable representation of the software design as it evolves.

3. Automated Requirements Tools - Computer programs which are used to provide succinct and unambiguous specification of the system based on computer requirements.

4. HIPO Design Aides - A graphical technique that defines each system component by the transformation of its input datasets to its output datasets and also defines the hierarchical relationships between components.

5. Process Design Language - A formal algorithmic specification of a software component.

6. Structure Charts - A graphical technique which illustrates the relationships between the components of a software system.

7. Top-Down Development - A software development approach that identifies major functions to be accomplished, then proceeds from there to an identification of the lesser functions that derive from the major ones.

8. Modular Decomposition - The process of breaking a large program into small modules that perform complete functions.

9. Program Support Library - A software system which provides tools to organize, implement and control software development.

10. Simulation - A computer program that provides the target system with inputs or responses that resemble those that have been provided by the process for the device being simulated.

11. Structured Programming - The activity of programming with a limited set of program constructs.

12. Walk-throughs - A formal meeting by various numbers of a project for the review of source code and design for technical adequacy and error detection.

13. Critical Piece First - The implementation of the most critical aspects of the system first.

14. Database Analyzer - A computer program that reports information on every usage of data, identifies each program using any data elements and indicates whether the program inputs, uses, modifies or outputs the data element.

15. Data Dictionary - A listing of the names, lengths and representations of all data items used in a software system. This tool may be manual or automated.

16. Documentation Generator - A computer program used to show in detail the logical structure of a computer program, usually by producing flowcharts.

17. High Order Language - A full repertoire of programming instructions and statements having formal syntax and lexical rules, usable in composing machine-independent source programs.

18. Pre-compiler - A computer program used to add capabilities to a system, as implemented by a language processor, that provides special-purpose features not normally included as part of its input.

19. Reusable Code - Source code originally developed for another application that can be used for a different application.

20. Correctness Proofs - The technique of proving mathematically that a program is consistent with a set of specifications.

21. Code Standards Auditor - A computer program used to automatically determine whether prescribed programming standards and practices have been followed.

22. Consistency Checker - A computer program used to determine (1) if requirements and/or designs specified for computer programs are consistent with each other and (2) if they are complete.

23. Independent Test Team - A project group not associated with the software design/development section which is responsible for testing software to check its compliance to specifications.

24. Program Flow Analyzer - A computer program that provides statistics on source code statement usage and timing data on program elements during test case executions.

END

FILMED

9-83

DTIC